VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Prostředí pro centralizovaný návrh a distribuci filtrovacích pravidel pro LAN sítě

Centralized Environment for Design and Distribution of Firewall Rules in Local Area Networks

2014 Jan Bednář

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Jan Bednář**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Prostředí pro centralizovaný návrh a distribuci filtrovacích pravidel pro LAN sítě
Centralized Environment for Design and Distribution of Firewall Rules in Local Area Networks

Zásady pro vypracování:

Cílem práce je navrhnout rozšiřitelné prostředí, které bude umožňovat ve vhodné vizuální podobě definici a otestování filtrovacích pravidel (traffic shaping, firewalling, QoS, ...) v centralizované podobě a následnou distribuci pravidel na cílová zařízení. V rámci práce bude navrženo API, umožňující přidání dalších typů zařízení. Funkčnost řešení bude otestována v implementaci nastavování pravidel v prostředí Linuxu (iptables) popř. Mikrotik RouterOS.

1. Prostudujte možnosti vizualizace a definice rozšiřujících pravidel pro firewalling, které nabízejí současné aplikace.
2. Zvolte implementační prostředí, které bude umožňovat snadný návrh pravidel a multiplatformnost řešení.
3. Proveďte důkladnou analýzu a návrh řešení, definujte mechanismus, kterým bude možno rozšířit prostředí o další moduly pro nastavení dodatečných filtrovacích pravidel a typů zařízení.
4. Naimplementujte základní jádro systému a jeho moduly pro firewalling a traffic shaping s exportními moduly pro iptables a RouterOS.
5. Vytvořte sadu několika ukázkových konfigurací, výsledné řešení otestujte a vyhodnoťte dosažené výsledky.

Seznam doporučené odborné literatury:

[1] BENVENUTI, Christian. Understanding Linux network internals. USA: O Reilly Media, Inc., 2006, 1035 s. ISBN 9780596002558
[2] PURDY, Gregor N. Linux iptables Pocket Reference: Firewalls, NAT & Accounting. USA: OReilly Media, 2004, 96 s. ISBN 978-0-596-00569-6.
[3] RASH, Michael. Linux Firewalls: Attack Detection and Response with iptables, psad, and fwsnort. USA: No Starch Press, 2007, 337 s. ISBN 978-1-59327-141-1.
[4] STALLINGS, William. SNMP, SNMPv2, SNMPv3, and RMON 1 and 2. 3, illustrated. USA: Addison-Wesley, 1999, 619 s. ISBN 9780201485349.
[5] Firewall. MikroTik Wiki [online]. [cit. 2012-06-14]. Dostupné z: http://wiki.mikrotik.com/wiki/Firewall

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Pavel Moravec, Ph.D.**

Datum zadání: 16.11.2012

Datum odevzdání: 07.05.2014

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2014 .....

Rád bych na tomto místě poděkovala všem, kteří mi s prací pomohli, protože bez nich by tato práce nevznikla.

Abstrakt

Diplomová práce se zabývá problémem správy síťových prvků, převážně směrovačů. Popisuje možnosti pro centralizovanou správu počítačové sítě a její vizuální reprezentaci. V diplomové práci je navrhnuto řešení nezávislé na operačním systému, které je modulární a je možno jej v budoucnu doplnit o podporu dalších typů zařízení a služeb. Je rovněž popsáno a implementováno univerzální webové rozhraní pro komfortní správu prvků sítě.

Práce se postupně zabývá rozborem dostupných technologií, současnou situací, snaží se analyzovat problém komunikace a nastavování síťových prvků, ukazuje řešení v podobě vlastního protokolu, který je následně implementován a testován.

Klíčová slova: Mikrotik, Python, JavaScript, Dojo Toolkit, TLS

Abstract

The master thesis deals with issues of network management and network elements, mostly about routers problems. The thesis describes possibilities for centralized management of computer network and its graphical visualization. In master thesis has been designed solution which is independent of used operating system, it can be modular and supports new features which can be implemeted in future. It is also described a implemented like a universal web interface for convenient management of network elements.

Thesis focuses on the analysis of available technologies, current situation, trying to analyze the main problem of communication and settings of network elements. These solutions have been implemented in the form of custom protocol a subsequently tested.

Keywords: Mikrotik, Python, JavaScript, Dojo Toolkit, TLS

Seznam použitých zkratek a symbolů

SSL	–	Secure Socket Layer
TLS	–	Transport Layer Security
GUI	–	Graphical User Interface
HTML	–	Hyper Text Markup Language
HTTP	–	Hyper Text Transfer Protocol
JSON	–	JavaScript Object Notation
URI	–	Uniform Resource Identifier
API	–	Application Programming Internface

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

V dnešní době lidé po celém světě využívají prakticky ve všech životních situacíh datové sítě. Stejně tak jsou moderní technologie, které využívají sítí součásti naších životů od úlého mládí. Již v predškolním věku dnes děti sedí u internetu, využívají tablety a chytré telefony. Následně jako studenti čerpají vědomosti a hledejí alternativní prameny právě na internetu. V produktivním věku si jen málo kdo umí přestavit, že by v práci fungoval bez počítačových sítí.

O to důležitější roli mají sítě v komerčním světě. Banky pomocí internetu převádí všemožné finanční částky. Úřady a firmy komunikují pomocí svých informačních systémů a elektronické pošty, skrze veřejné i soukromé komunikační kanály. Všechna odvětví průmyslu, obchodu i běžného života jsou neodmyslytelně provázány různými typy počítačových sítí.

Každý uživatel si přeje, aby mu síť fungovala a nemusel se o ni starat. Podobná situace je na straně poskyovatelů internetových a datových služeb. Snaží se co nejvíce úkonů zjednodušit a automatizovat. Od fakturací po nastavení koncových bodů sítě. A právě o tomto problému pojednává tato diplomová práce.

Cílem této práce je navrhnutí protokolu pro jednotné nastavení všech uzlů datové sítě, jeho implementace a reálné použití v praxi.

První část se zabývá vysvětlením teorie o nástrojích a principech užitých k vytvoření funkční implementace, která řeší zadaný problém. Jsou zmíněny technologie přenosů, jejich formát a programovací jazyky a nástroje vedoucí k funkčnímu celku.

Druhá část popisuje současnou situaci na poli centralizované správy a vizualizací sítě.

Třetí část rozebírá problematiku a nastiňuje možné řešení.

V další je zmíněno, že je potřeba navrhnout způsob komunikae. Nejprve jsou popsány požadavky, které jsou kladeny na vlastnosti protokolu, poté je zmíněno řešení a na konec navrhnuty reálné postupy, hodnoty a reprezentace dat.

Pátá část se zabývá použitím navrhnutého řešení do praxe. Popisuje Jednotlivé části systému a jejich funkce.

Poslední část je věnována nastavení a testování reálné ukázky.

2 Teoretický rozbor

2.1 Secure Sockets Layer

SSL (Secure Sockets Layer) je nekomerční otevřený protokol a v současné době jedna z nejvíce používaných metod pro zabezpečení datových přenosů v rámci internetu mezi serverem, například s webovou prezentací, a uživatelem. Překlad názvu je *vrstva bezečných socketů* a vjadřuje pravou podstatu protokolu. Pracuje mezi transportní a aplikační vrstvou ISO/OSI modelu

2.1.1 Certifikační autorita

Certifikační autorita (CA) je instituce, která vydává a ověřuje digitální certifikáty. Podobně jako např. státem autorizovaný notář, je také certifikační autorita zodpovědná za ověření skutečné identity nositele certifikátu.

Každý žadatel o přidělení certifikátu je povinnen sdělit informace o sobě, které jsou následně ověřeny pomocí ověřeného výpisu z obchodního rejstříku, občanského průkazu, či majitele internetové domény.

2.1.2 Certifikáty

Jedná se o veřejný klíč, který vydává certifikační autorita. Certifikát mimo nejdůležitější položky - veřejného klíče - obsahuje také informace o majiteli a certifikační autoritě, která ho vydala.

Aby se předešlo podvržení nepravého certifikátu nepravou cerfitikační autoritou, je-li to možné, ověřuje klient pravost certifikátu přímo u certifikační autority, která dotyčný certifikát vydalala.

Každý certifikát obsahuje následující údaje:

- seriové číslo - identifikátor certifikátu
- alforitmus použitý pro vytvoření podpisu
- informace o majiteli certifikátu
- informace o vydavateli certifikátu
- časové rozmezí platnosti
- účel, pro který je certifikát vytvořen
- specifikace algoritmu pro otisk certifikátu
- otisk certifikátu sloužící jako kontrolní součet správnosti dat
- veřejný klíč

2.1.3 Fáze komunikace

2.1.3.1 Dohoda algoritmů

1. Klient zašle inicializační zprávu, ve které oznámí nejvyšší podporovanou verzi a dostupné šifrovací algoritmy.
2. Server odpoví také inicializační zprávou, oznámí verzi protokolu a vybere jeden konkrétní šifrovací algoritmus ze seznamu, který poslal klient
3. Pokud to zvolená šifra umožnujě, zašle server klientovi svůj certifikát. Je-li to nutné, může také požádat klienta o zaslání jeho cerfitikátu.

2.1.3.2 Výměna a nastavení klíčů

1. Pokud to zvolený šifrovací algoritmus umožnujě, zašle klient veřejný klíč.
2. Obě strany vytvoří symetrický šifrovací klíč podle předem sdělených informací.

2.1.3.3 Vlastní komunikace Pokud kterýkoli z předcházejících kroku skončil nezdarem, je vyhodnoceno spojení jako nezabezpečené a následně ukončeno. Je-li nastavení klíčů úspěšné, lze nyní přistoupit k vlastní komunikaci, pomocí symetrického klíče, který byl vygenerován v předchozím kroku. Tento klíč je používán až do ukončení spojení.
[2]

2.2 RouterOS

Mikrotik RouterOS je operační systém vyvíjený od roku 1995 firmou Mikrotik. Systém je založený na linuxu a upravený pro potřeby aktivních síťových prvků, v drtiné většině směrovačů. Lze jej spustit na mnoha platformách, včetně tzv. `embedded` platform určených pro specifické aplikace, v tomto případě aplikace spravující provoz datových sítí.

Konfiguraci systému lze provést více způsoby. Mezi asi nejpoužívanější patří konfigurace pomocí vlastní dodávané utility WinBox pro OS Windows, která velice připojíná prostřední opračního systému od firmy Microsoft. Z příkazové řádky je dostupná konzole TELNET, případně její šifrovaná obdoba SSH.

Dostupné možnosti konfigurace a komunikace s RouterOS:

- **telnet** - Základní konzolový protokol poskytující nešifrovaný přenos dat v podobě čistého textu. Obsahuje vlastní rozhraní CLI. Naslouhcá standardně na portu 23.
- **ssh** - Zabezpečený konzolový protokol podporující večejné klíče pro šifrování spojení. Ve zbytku je práce s ním v RouterOS stejná, jako s Telnetem. Naslouchá na portu 22.
- **ftp** - Protokol pro přenos souborů. Využívá nezabezpečené komunikace a běží na portu 20

- **winbox** - Windows utilita pro správu, která pomocí přehledného grafického rozhraní je schopna pracovat s nastavením a čtením hodnot a navíc také přenášet soubory. Využívá zabezpečené spojení na portu 8291.
- **www** - Webové rozhraní, které je aktuálně vybaveno vlastním GUI s názvem WebFig. Je velice podobné utilitě winbox. Využívá protokol HTTP na portu 80.
- **www-ssl** - Webové rozhraní zůstává stejné, jako u `www`, rozdíl je však v komunikaci, kdy je zde narozdíl od čistého textu přenášeného u www využito šifrování SSL. Standardní port je 443, ale v základním nastavení je tento protokol HTTPS vypnut.
- **api** - Vlastní protokol pro komunikaci se zařízením. Data jsou přenášena nezabezpečeně na portu 8728. Presnější popis tohoto rozhraní je uveden dále v textu.
- **api-ssl** - Obdoba protokolu `api`. Formát dat, syntaxe, požadavky i odpovědi jsou totožné. Liší se v zabezpečení, kdy využívá šifrování SSL, resp TSL.

2.2.1 API

RouterOS je pro komunikaci s jinými automatizovanými systémy vybaven vlastním API, takzvaným `MikrotikAPI`. Používá protokol TCP a standardně je dostupné na portu 8728 a od verze RouterOS v6.1 na portu 8729 s podporou SSL šifrování.

Příkazy pro API vychází ze CLI příkazů. Můžeme tedy počítat s tím, že existuje-li ve CLI `/nejaka/vlastni/cesta`, bude tato sekvece dostupná také v API. Neexistuje bohužel žádná mapa, resp. seznam dostupných příkazů, jako je tomu například v MIB u SNMP.

2.2.2 Slovo

Základní informací při použití API je takzvané `slovo` (WORD). Jedná se prakticky o reprezentaci textového řádku do podoby:

velikost slova	obsah slova

Tabulka 1: Definice slova

Obsah slova lze rozdělit do následujících sekcí:

- **příkaz** - začíná vždy lomítkem a je prvním slovem každé `věty`. Můžeme tedy říci, že příkaz je prvním slovem každé věty. Definuje, zda-li cheme číst, zepisovat, případně vykonávat jiný speciální příkaz a o jakou hodnotu se bude jednat. Pro výpis všech IP adres použijem `/ip/address/getall`
- **atribut** - začíná rovnítkem a nastavuje hodnotu, kterou obsahuje. Atribut určující ip adresu: `=address=10.0.0.1`
- **API atribut** - začína tečkou a v současné době podporuje jen atribut `tag`, tedy kupříkladu `.tag=2`

- **dotaz** - uvozuje jej otazník. Nastavuje se jím vyhledávací filtr v dostupných položkách. Pokud budeme chtít vyfiltrovat pouze rozhraní typu vlan: `?type=vlan`
- **odpověď** - prvním znakem je vykřičník a znamená typ odpovědi na některý z příkazů. Každá věta, která je odpovětí na dotaz obsahuje alespoň jednu odpověď
  - `!re` - informuje, že následuje výpis hodnot
  - `!done` - konec odpovědi (věty)
  - `!trap` - signalizuje chybu

2.2.3 Věta

Věta je definována posloupností slov. V praxi vypadá jako blok řádků(slov), který začíná příkazem, můžeo obsahovat několik atributů a doazů a je zakončen prázdným slovem, tedy řádkem.

2.3 Python

Jazyk Python začal vznikat v roce 1989 ve výzkumném ústavu v Amsterodamu. Za zakladatele tohoto skruptovacího jazyka je považován Guido van Rossum a je patrné, že u samotného návrhu dostatečně přemýšlel. Vznikl promyšlený jazyk, který je však stále ve vývoji. Jeho jméno je odvozeno od pořadu BBC Monty Python's Flying circus. V současnosti je možno tento jazyk provozovat na mnoha platformách (Linux, Win, Mac, WinCE, OS2, Java). Stejně tak programy a skripty v něm napsané lze na těchto systémech téměř vždy spouštět bez úprav.

A jaký Python vlastně je? Čistý objektový jazyk se správou výjimek, kompilací do bytecodu, mnoha vysokoúrovňovými typy (řetězce, seznam, asociativní pole). Vývoj se rozdělil na dvě větve. Ta jedna podporuje dodnes nejčastěji používanou verzi 2 a ta druhá pracuje na vývoji verze s označením Python 3, plně podporující Unicode. Python lze doplnit o vlastní vestavěné typy a funkce pomocí C/C++, nebo naopak lze interpret začlenit do programu v jiném jazyce. V základu obsahuje velké množství modulů, které lze ihned používat. Mezi důležité moduly patří například moduly pro přístup k databázím, GUI, službám operačního systému, HTTP, FTP, POP3, SMTP a mnohým jiným protokolům. Knihovna pro regulární výrazy se stala také pevnou součástí jazyka. Jsou nadefinovány také moduly pro přístup k vnitřním mechanismům Pythonu (garbage collector, parser, kompiler). V Pythonu je taktéž napsán debugger a profiler tohoto jazyka. [1]

Python je dnes používán na serverech po celém světě, nejčastěji ve verzíh 2.6 a 2.7. Verze 3 se vyskytuje spíše sporadicky. Pravděpodobně je to způsobeno zpětnou nekompaktibilitou k verzi 2.x. Nelze tedy spouštět staré skripty přímo v novější verzi. Existují mechanizmy a skripty pro převod Python 2 na Python 3, využívá je ale minimum lidí, protože verze 2.7 poskytuje všechny potřebné knihovny a možnosti zápisu. Hlavní výhodou Python 3 je nativní podpora znakové sady unicode, jak již bylo zmíněno dříve.

Seznam Python hostingů, které jsou zdrama lze najít na http://wiki.python.org/moin/PythonHosting. Z komerčních můžeme zmínit například českého provozovatele hostingu Roští.cz.

Poznámka 2.1 Pokud chceme z novější verze Python 3 používat některé vylepšení, je do starší verze zahrnuta knihovna `__future__`, ze které lze naimportovat například použití `print` jako funkce, případně nejvíce diskutované použití znakové sady Unicode jako výchozí (`unicode_literals`. Celý zápis vypadá následovně:

```
#do A i B vlozime stejny text
a="aa"
from __future__ import unicode_literals
b="bb"
#vysledky (a == 'aa', ale b == u'bb')
type(a)
<type 'str'>
type(b)
<type 'unicode'>
```

Výpis 1: Importování unicode znakové sady

Python pro definici bloků nepoužívá tak časté složené závorky, ale odsazeni rádků zleva. Je jedno, zda-li se jedná o tabulátory, nebo mezery a kolik jich je potřeba k uvození bloku. Je pouze důležité zachovat v celém souboru stejné podmínky. Na každém řádku je právě jeden příkaz a je volitelně zakončen středníkem. Obecné definice a doporučení jsou vydávány pod označením PEP a jsou dostupné na webu http://legacy.python.org/dev/peps/.

2.4 JavaScript

JavaScript je objektově orientovaný skriptovací jazyk, která vznikl ve společnosti Netscape v roce 1995. Jeho zápis je bližší spíše C/C++, než jazyku Java, se kterým má z marketingových důvodů pouze společný název.

Standard jazyka byl v roce 1997 publikován organizací ECMA (European Computer Manufacturers Association) a podle ní se také tato standardizovaná verze nazývá ECMAScript. Všechny další implementace jazyka následně vychází z této verze.

Dnes se JavaScript využívá primárně jako doplněk internetového prohlížeče WWW stránek na straně klienta, vkládaný přímo do HTML souborů. Takto lze zajistit dynamicky se měnící obsah, případně vytvořit grafické rozhraní GUI, aby se stránka co nejvíce přiblížila stolním aplikacím. Každý webový prohlížeč implementuje vlastní verzi JavaScriptu, který však vždy musí vycházet z ECMAScriptu.

2.5 JSON

JSON je zkratka pro JavaScript Object Notation, tedy způsob zápisu objektů, který se poprvé objevil ve stejnojmenné jazyce. Jedná se tedy o podmnožinu programovacího jazyka JavaScript podle standardu `ECMA-262` revidovaného v prosinci 1999.

Pro své prosté a univerzální řešení se formát rozšířil i do ostatních jazyků. JSON využívá jednoduchý textový zápis, který je zcela nezávislý na jazyce. Umožňuje zapsání čísel, textů, polí, slovníků i objektů. Je tedy idální k jednoduché serializaci dat a objektů do textového zápisu. JSON nedefinuje kódování znakové sady. Většina užívaných implementací v programovacích jazycích užívá jako výchozí znakovou sadu Unicode UTF-8.

Ve srovnání s jinými způsoby zápisu dat neobsahuje žádné definice kontextu dat. Veřejně se udává, že je tedy přibližně o 40% úspornější na zápis, než například jazyk XML. Oproti tomuto zápisu však nedefinuje ve svém základu mechanizmy pro vyhledávání v datech. V poslední době se proto ujímá standard JSON Pointer, definovaný v RFC 6901. Jde o zjednodušenou náhradu výrazů XPath.[4]

```
{
   "0":1,
   "1":−2,
   "2":3.333,
   "3":4.0e+17,
   "4":"abc",
   "5":"\u00e1\n",
   "6": null ,
   "7" :[2.1,2.2,[ "2.2.1" ]],
   "8": false ,
   "9": true ,
   "10":"",
   "key":"value",
   "abc\"def" :[]
}
```

Výpis 2: Zápis indexovaného pole různých objektů

Jedná se o implicitní zápis objektů jazyka JavaScript, takže v tomto jazyce se načtení zápisu řeší povětšinou funkcí `eval()`, která pouze zápis spustí.

Pro prakticky všechny možné jazyky již existují moduly pro převod tohoto formátu. Některé jazyky mají tyto moduly již v základu implementovány a přibaleny. Jedná se například o Python, PHP, ASP, Java a další.

2.6 RESTful

Representational State Transfer je architektura primárně pro webové API. Publikoval ji v roce 2000 jeden ze spoluautorů protokolu HTTP Roy Fielding. Fielding používá pro komunikaci základní příkazy i odpovědi protokolu HTTP verze 1.1 tak, jak jsou popsány v RFC 2616.

REST je orientován datově, určuje tedy pouze přístup k datům a nikoli jejih reprezentaci, kterou přenechává na vyšší vrstvy.

Základní vlastnosti:

- **klient-server** - Data jsou uložena na serveru a klient pouze zasílá požadavky
- **bezstavový** - Každý požadavek musí fungovat jako samotný. Musí tedy obsahovat všechny náloežitosti pro jeho vykonání bez ohledu na předcházející požadavky
- **cache** - U požadavku může být nastaveno, zda-li se má kešovat, či nikoli
- **CRUD** - 4 základní metody: Create, Retrieve, Update a Delete, které musí vystačit pro komunikaci. Jelikož jsou totožné s protokolem HTTP, jsou jím implementovány.

- **reprezentace dat** - Data jsou uložena v těle požadavku, pro komunikaci je tedy jedno, jaký používají formát reprezentace dat. Doporučeny jsou ATOM/RSS, JSON a XML.

2.6.1 URI

Takzvaný jednotný identifikátor zdroje, reprezentovaný textovým řetězem a využívaný zejména na internetu, kde jej všihni znají z cesty k html stránkám, jako například `http://www.vsb.cz/st`
Tato cesta má následující předpis:

```
schéma:hierarchická cesta?dotaz#umístění
```

s takovýmto významem jednotlivých částí, které jsou odděleny speciálními znaky:

- **schéma** - druh adresy (mailto, ftp, https, ...)
- **hierarchická část** - popis jednoznačné cesty k dokumentu
- **dotaz** - blíže specifiguje adresu (volitelné)
- **umístění** - určuje o kterou část dokumentu se jedná (volitelné)

Pro náš případ je podstatná pouze část hierarchické cesty, která se dále zúží o adresu zařízení. Zůstane tedy `/stranka`
V případě architektury REST je možné zasílat požadavky na následující cesty:

- `/` - dotaz na celou databázi
- `/stranka/` - dotaz na kolekci, resp tabulku v případě databáze, s názvem `stranka`
- `/stranka/id` - dotaz na konkrétní dokument s unikátním identifikátorem `id`

2.6.2 Požadavky

Celá architektura definuje pouze 4 typy požadavků: Create, Retrieve, Update, Delete, které odpovídají HTTP požadavkům POST, GET, PUT a DELETE.

	/tabulka/	**/tabulka/id**
GET	vrátí seznam objektů	vrátí objekt
POST	vytvoří objekt	vytvoří objekt
PUT		upraví objekt
DELETE		odstraní objekt

Tabulka 2: Interpretace dotazů na různá URI

Celý příkaz se vždy skládá z požadavku, který je následován parametrem ve tvaru URI. Jako oddělovaš louží obyčejná mezera.

2.6.2.1 GET (Retrieve) Příkaz slouží k získání obsahů dokumentů ze serveru.

Například požadavek `GET /uzivatel/4` vrátí jako odpověd informace u uživateli s identifikátorem 4, pokud je takovýto identifikátor nalezen.

Možné dotazy na různá URI:

- **http://server/** - Získá seznam kolekcí dostupnýh na serveru.
- **http://server/kolekce/** - Vrátí seznam dokumentů v kolekci
- **http://server/kolekce/id** - Načte a vrátí konkrétní dokument

Možné odpovědi:

- **200 OK** - Požadovaný dokument byl nalezen a předán na výstup.
- **404 Not Found** - Požadovaný dokument nebyl na serveru nalezen nalezen.

2.6.2.2 POST (Create) Pokud je potřeba vytvořit nový dokument, případně kolekci, zašle se příkaz s URI nadřazené kolekce.

Například požadavek `POST /uzivatel/` vytvoží nový záznam v tabulce uzivatel. Identifikátor záznamu se nepoužívá, protože v době zápisu není klientovi znám.

Možné dotazy na různá URI:

- **http://server/** - Vytvoří novou kolekci.
- **http://server/kolekce/** - Vytvoří nový dokument v požadované kolekci.

Možné odpovědi:

- **200 OK** - Obecná informace o tom, že byl příkaz vykonán.
- **201 Created** - Dokument byl vytvořen. V těle odpovědi je možné specifikovat nově vytvořený identifikátor právě vytvořeného objektu
- **204 No Content** - Dokument byl vytvořen, ale odpověď o něm neobsahuje žádné informace.
- **404 Not Found** - Dokument nebylo možné vytvořit.

2.6.2.3 PUT (Update) O editaci záznamů se stará příkaz PUT. Je velice podobný příkazu POST. V době editace je oproti předchozímu příkazu však již znám identifikátor, který tedy musí být specifikován.

Například požadavek `PUT /uzivatel/4` přepíše záznam o uživateli s identifikátorem 4.

V případě, že chce klient rozhodnout a nastavit přesně identifikátor nově vkládaného objektu, lze příkaz použít také k vytvoření.

Možné dotazy na různá URI:

- **http://server/kolekce/** - Upraví nastavení kolekce
- **http://server/kolekce/id** - V případě, že objekt existuje, upraví jej. Pokud objekt naopak neexistuje, pokusí se vytvořit jej a nastavit mu požadovaný identifikátor.

Možné odpovědi:

- **200 OK** - Obecná informace o tom, že byl příkaz vykonán.
- **201 Created** - Pokud byl příkazem vytvořen nový dokument.
- **204 No Content** - Dokument byl upraven, ale odpověď o něm neobsahuje žádné informace.
- **404 Not Found** - Dokument nebylo možné upravit. Většinou nebyl specifikován identifikátor.

2.6.2.4 DELETE (Delete)

Objekt lze smazat pomocí příkazu DELETE. Pro smazání objektu je potřeba v URI nastavit přesný identifikátor. Pokud takovýto objekt nebude nalezen, bude vrácena chyba. Chyba bude vrácena taktéž při druhém volání, pokud zavoláme stejný příkaz dvakrát po sobě.

Například požadavek `DELETE /uzivatel/4` smaže záznam o uživateli s identifikátorem 4.

Možné dotazy na různá URI:

- **http://server/kolekce/id** - V případě, že objekt existuje tak jej odstraní.

Možné odpovědi:

- **200 OK** - Obecná informace o tom, že byl příkaz vykonán.
- **204 No Content** - Dokument byl smazán, ale odpověď o něm neobsahuje žádné informace.
- **404 Not Found** - Dokument nebylo možné smazat, protože neexistuje.

[3] [5] [11] [10] [8]

2.7 SQLite

SQLite je jednoduchý databázový systém, který je dostupný pouze lokálně a pro uložení všech dat využívá jediný soubor.

2.7.1 Datové typy

SQLite databáze je takzvaně beztypová. Znamená to, že je prakticky jedno, jaký datový typ uložíme do kterého sloupce. Může se tedy stát, že deklarujeme sloupci číselný datový typ a uložíme do něj textový řetězec.

Jedinou vyjímkou je sloupec s typem INTEGER PRTIMARY KEY, kdy musí bát použito celé číslo z důvodu rychlého indexování,

Každá uložená hodnota v databázi je uložena do jedné z následujících typových tříd:

- **NULL** - hodnota nemá žádnou hodnotu
- **INTERGER** - celočíselná hodnota, která je uložena v 1 až 8 bajtech, v závislosti na velikosti čísla.
- **REAL** - číslo s plovoucí desetinou řárkou
- **TEXT** - textový řetězec, který je uložen ve standardním formátu databáze (UTF-8, UTF-16BE nebo UTF-16LE)
- **BLOB** - binární data, která jsou uložena ve stejném tvrau, jako byla vložena na vstup

Pro uložení datového typu Boolean není v databázi definována žádná třída. Využívá se typu INTEGER, kdy je hodnota False zapsána jako nula a hodnota True je zapsána jako číslovka jedna.

Stejně tak není exaktně určeno, jak se budou ukládat informace o datumech a času. Jednou z variant je použití typu TEXT, do kterého se uloží datum ve tvaru, který definuje norma ISO8601, tedy `YYYY-mm-dd HH:MM:SS.SSS`. Další variantou je použití třídy INTEGER a uložení do formátu takzvaného Unix Timestamp, tedy počet vteřin od počátku počítačového věku, který byl určen jako půlnoc prvního ledna roku 1970.[9]

2.7.2 Omezení

Databáze podporuje drtivou většinu funkcní podle standardu SQL-92. Hlavní nevýhodou oproti klasickým databázím je obsence uživatelů a nastavení jednotlivých práv přístupu.

Celková velikost databáze je omezena velikostí $2^{41} bajt, coodpovdavelikostidvaterabajty. Toholzevpraxi,$

Nejhorší omezení, kterého lze navíc v praxi realně dosáhnout je velikost jenoho řádku. Tato hodnota je stanovena na jeden megabajt.[7]

3 Současná situace

V současnosti nebyl nalezen žádný systém, který by alespoň částečným způsobem vyhovoval zadání.

Jsou k dispozici buď to systémy, ve kterých je možné vymodelovat základní podobu počítačové sítě, avšak tyto systémy nemají možnost definice dat jednotlivých prvků sítě.

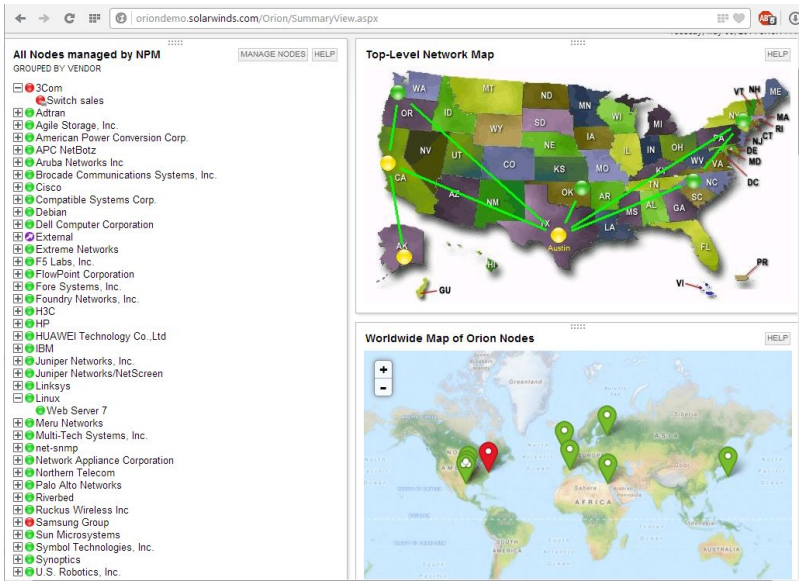
Na druhé straně existují systémy podporující centralizované nastavení síťových prvků, které nemají možnost zobrazit žádný model sítě.

3.1 Obdobné nástroje

3.1.1 Solarwinds Network Configuration Manager

Severoamerická společnost Solarwinds sídlící v Texasu nabízí produkt s názvem **Network Configuration Manager**. Jeho cena začíná na ceně zhruba okolo 60 000 Kč.

Aplikace pro svůj běh potřebuje minimální konfiguraci s vysokými nároky na paramatry serveru, jak jsou dvoujádrový 3GHz procesor a 3GB operační paměti. Ke svému spuštění vyžaduje operační systém Microsoft Windows 2003 SP2, nebo novější, .NET Framework 4.0. Data jsou ukládána na SQL server minimálně verze 2005, vyvíjený rovněž firmou Microsoft a označený MSSQL.



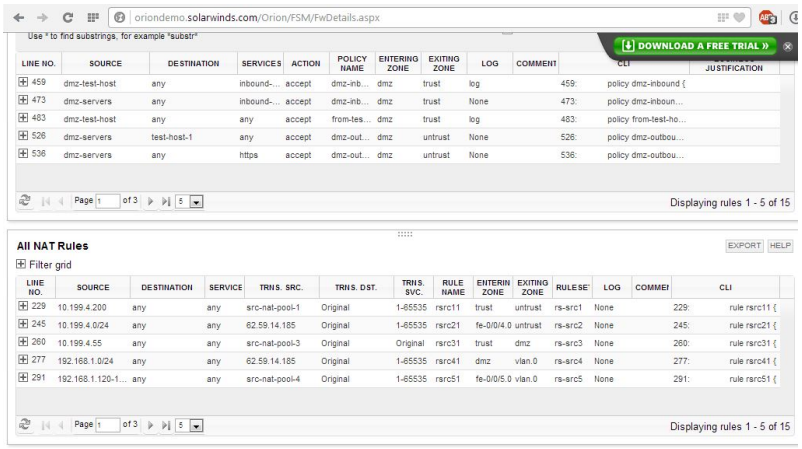
Obrázek 1: Grafické zobrazení sítě

Software společnosti Solarwinds je pravděpodobně zaměřen na sféru národních a nadnárodních korporací. Tomu je přizpůsoben i seznam zařízení, které systém podporuje. Jedná se například o:

- Cisco

- HP
- Juniper
- Dell
- Linksys
- Synoptics
- Palo Alto Networks
- a další podobní výrobci...

Najdeme zde tedy podporu pouze high-end síťových prvků. Námi potřebný Mikrotik RouterOS zde bohužel nenajdeme a nemůžeme se za to zlobit, jelikož zařízení tohoto výrobce jsou určená pro naprosto jiný segment trhu.



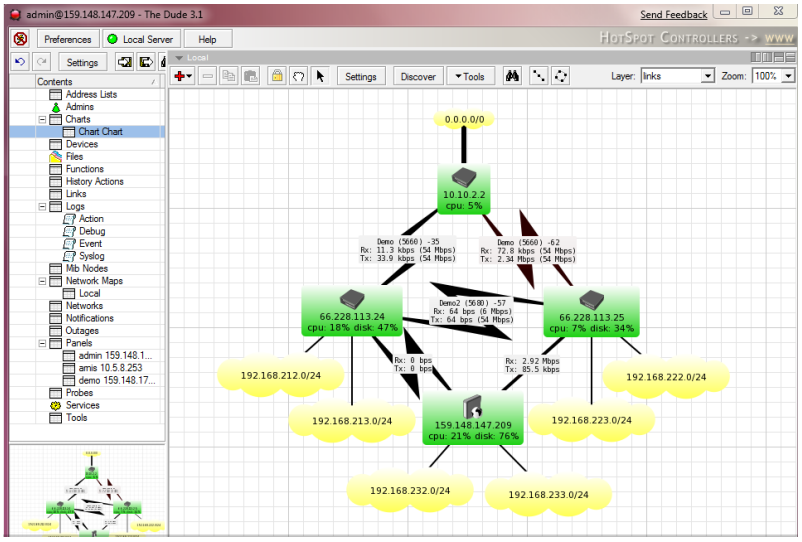
Obrázek 2: Definice pravidel firwallu

Nastavení firewallu je zde propracováno výborně. Chybí zase ale ostatní potřebné moduly, pro omezení síťového provozu skrz zařízení nebo filtrování fyzických adres.

3.1.2 Mikrotik Dude

Dude je monitorovací a dohledový systém vyvíjený společností Mikrotik. Pomocí tohoto software lze monitorovat a spravovat všechny zařízení společnosti Mikrotik.

Definice sítě je řešena vytvořením modelu sítě jako diagramu v grafickém prostředí. Lze nastavovat všechna pravidla, avšak pouze v reálném čase.

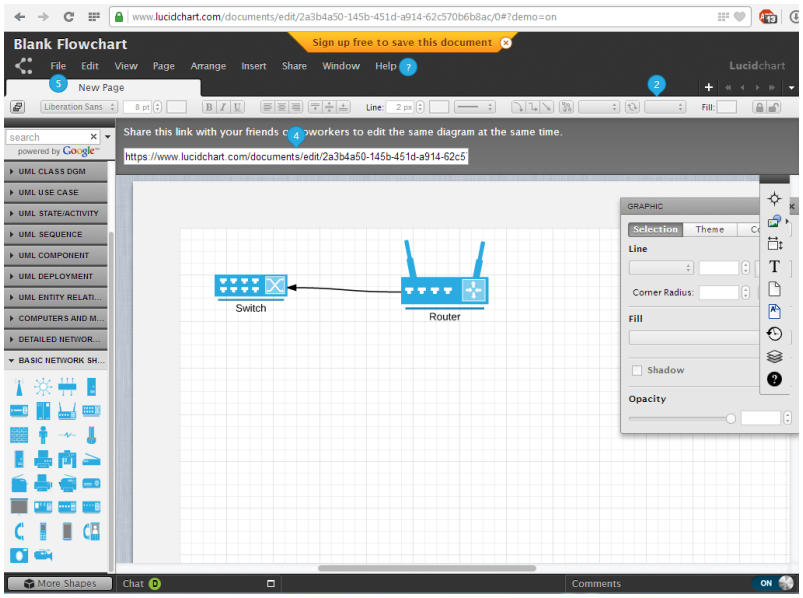


Obrázek 3: Vzhled nástroje Dude

3.2 Vizuální modely

Další sekci utilit tvoří online nástroje pro modelování diagramů. Ty nemají s nastavením a informacemi o datových sítí nic společného. Řeší ale grafickou část reprezentace sítě.

Mezi takovéto povedené nástroje patří LucidChart.com, běžící ve Flashi, nebo draw.io, který je napsán v JavaScriptu.



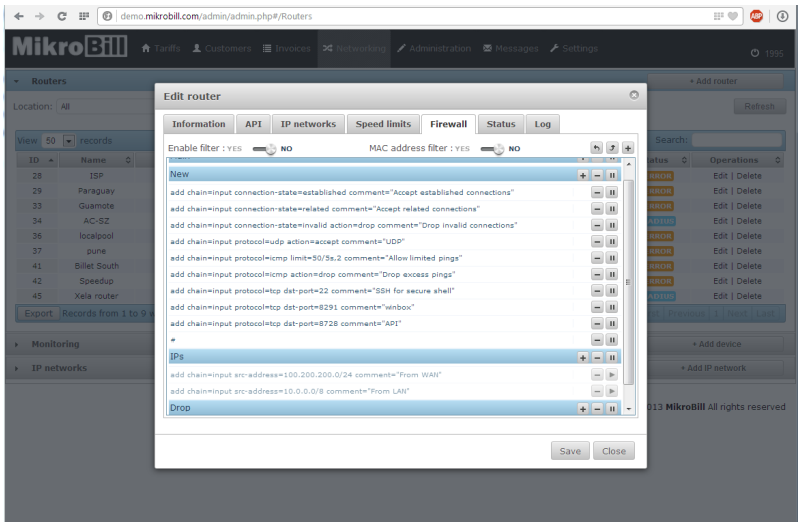
Obrázek 4: Nástroj Lucidchart pro kreslení diagramů sítě

3.3 Nástroje pro správu

Poslední část nástrojů lze shrnout jako vhodné pro správu a nastavení sítě, kterým však chybí grafické znázornění konfigurované sítě.

3.3.1 MikroBill

MikroBill je software české společnosti Neomax CZ. Systém nabízí správu zákazníků, automatikou fakturaci a v neposlední řadě také nastavení jednotlivých prvků sítě v příjemném webovém prostředí. Software podporuje jen MikrotikAPI a bohužel v jeho nezabezpečené podobě.



Obrázek 5: Konfigurace firewallu pomocí MikroBill

3.4 rConfig

Nástroj rConfig poskytuje správu zařízení na platformě Linux. Pro takovéto zařízení je schopen definovat určitou sadu pravidel a příkazů, pomocí kterých je schopen nastavit prakticky vše.

Jeho velikou nevýhodou je, že neumožňuje definovat nastavení zařízení jednotným způsobem a podporuje pouze Linuxu podobné systémy.

3.5 ISPAdmin

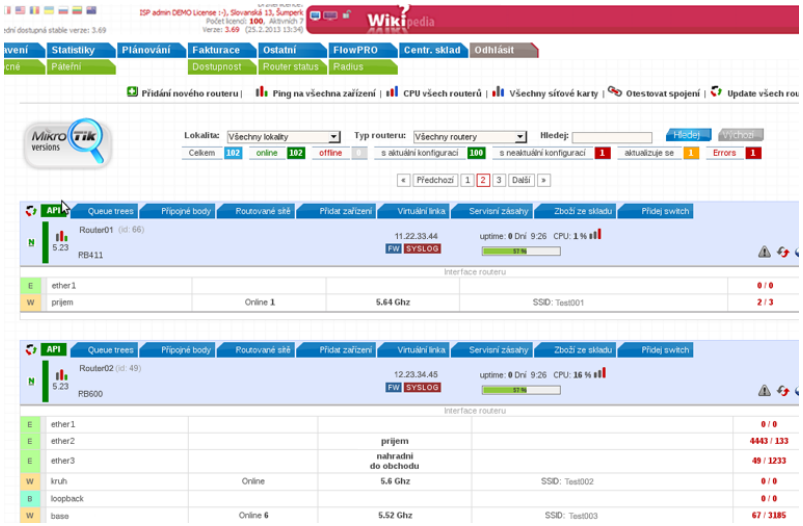
ISPAdmin je komplexní systém pro management větších sítí na úrovni poskytovatelů datového připojení.

Systém zvládá provádět základní nastavení pro zařízení 4 typů:

1. MikroTik

2. Ubiquity
3. MotorolaCanopy
4. Linux

V přidání dalších typů narazíme na problém, jelikož se jedná o proprietální komerční systém. Nutnost za systém platit ho znevýhodňuje oproti jiným. Není možné rovněž zobrazit podobu sítě. Tedy je zde možnost náhledu sítě v monitorovacím systému Nagios, ten však podporuje pouze hvězdiové sítě a v případě použití OSPF se stává nepoužitelným.



Obrázek 6: Seznam routerů v ISPAdmin

4 Analýza problému

V dnešní době není dostupný zádný nástroj, který by umožňoval zabezpečeně dopravovat konfigurace síťových služeb na koncová zařízení různých výrobců a používající nespočet různých systémů a byl přitom modulární a na tolik zdokumentovaný, aby bylo možné dopsat adaptéry na nová zařízení.

Pokud k tomu přídáme navíc nutnost celou síť zobrazit, nebyde nám jiná možnost, než navrhnout vlastní řešení.

Celý problém lze rozdelit na několik měnších částí, z nichž každá může být řešena do určité míry samostatně.

Problémy rozeberem v pořadí, ve kterém by měly být použity při běžné obsluze.

4.1 Definice sítě

Jak ze zadání vyplývá, je potřeba definovanou síť zobrazit takovým způsobem, který by zajistil, že úprava bude jednoduchá, centralizovaná a je možné do síťe přidávat i odebírat prvky na různých místech.

Ve vlastním řešení se to pokusíme implementovat pomocí diagramu sítě, který bude dynamický.

Pokud máme nadefinovanou síť tak, aby přibližně odpovídala skutečné topologii a máme nastaveny všechny adresy a informace o jednotlivých zařízeních, můžeme pristoupit k nastavení pravidel.

4.2 Úprava pravidel

Je potřeba, aby každý prvek sítě mohl být nastaven bez ohledu na výrobce a použitý systém. Mělo by být dosaženo jednotného rozhraní pro všechny zařízení v síti.

Jedná se primárně o pravidla firwallu, definice statických MAC adres a omezení rychlosti pro jednotlivé adresy.

Následně se pokusíme vytvořit v JavaScriptu uživatelské rozhraní, které umožní jednotně definovat potřebné nastavení.

4.3 Přenos pravidel na zařízení

Jednou z klíčových funkcí je zejména přenos vytvořené konfigurae na koncová zařízení, zvláště v situaci, kdy je každé zařízení vybaveno jiným systémem.

Celá komunikace navíc nesmí být čitelná při průchodu sítí od centrálního systému ke koncovému uzlu a nesmí být možné ji při přenosu pozměnit.

Navrhneme nejlépe vícevrstvá šifrování, které ve vhodném formátu přenese data mezi zařízeními. Nebude-li se z jakéhokoli důvodu na koncové zařízení možné připojit přímo, použijeme nejbližší zařízení, které to umožní a z tohoto mezičlánku nastavíme koncové zařízení jiným možným způsobem.

4.4 Aplikování pravidel

Jakmile data přesuneme na cílový prvek nastává problém s aplikováním nastavení. Různé systémy mohou mít různě řešené konfigurace a potřebné služby.

Ideální by bylo pro každé takové zařízení vytvořit jakýsi převodník mezi těmito formáty, tedy Konektor.

5 Vlastní komunikační protokol

Z důvodu zajištění komunikace mezi jednotlivými zařízeními a centrálním serverem bylo potřeba navrhnout protokol pro předávání potřebých dat.

5.1 Požadavky

Na samém počátku vývoje protokolu bylo potřeba stanovit určité podmínky, které musí protkol splňovat:

1. Data budou převáděna z nativních objektů daných programovacích jazyků. Musí být tedy pro přenos použit formát, který dovolí zapsat prakticky jakýkoli objekt.
2. Převaděč, který má na starost konverzi mezi objekty a transportním formátem bude možno nahradit jiným. Znamená to tedy, že formát přenášených dat může být kdykoli změněn bez dopadu na funkčnost protokolu.
3. Jednotlivé objekty, které se budou přenášet nemusí být v době vzniku protokolu ještě známy a je potřeba mít možnost je rozšířit o objekty nové.
4. Jelikož se budou data přenášet po rozsáhlých datových sítí, u kerých není zaručena bezpečnost, musí být protokol pro přenos odolný vůči odposlouchávání.
5. Není zaručeno, že nebudou data posílána skrze aktivní prvky třetích stran, které mohou být vybaveny různými formami proxy pro pozměnění dat v průbehu cesty (takzvaný útok Man-in-the-middle). Protokol tedy musí být odolný i vůči takovémuto útoku.

5.2 Řešení

Pro uspokojení požadavků byly navrženy dílčí řešení. Každé řešení odpovídá požadavku se stejným označením.

1. Doporučeno použití formátu JSON jako základního. Lze použít také ATOM/RSS, XML, nebo jiný vlastní zápis, který splňuje požadavky.
2. O takzvanou serializaci se musí starat externí knihovna, která ovšem podporuje předchozí bod.
3. Objekty jsou definovány jako moduly, které lze do systému přidávat.
4. Lze použít šifrování pomocí libovolného algoritmu, například AES. Díky následujícímu bodu řešení je tato část nepovinná.
5. Spojení bude sestaveno nad protokolem SSL/TLS při použití certifikátu s veřejným klíčem.

Z toho vyplývá, že náš protokol je plně modulární a pro sestavení spojení využívá socketu uvnitř šifrovací vrstvy SSL/TLS.

5.3 Datové typy

Pro potřeby námi definovaného protokolu je potřeba nadefinovat typy dat, se kterými budeme pracovat. Použité datové typy vychází ze základních datových typů jazyka Python. Lze je však analogicky strovnat s obdobnými typy v jiných programovaích jazycích.

1. **int** - celé číslo podporující záporné znaménko
2. **str** - textový řetězec s kódováním unicode ve verzi utf-8
3. **list(x)** - seznam, resp pole, jehož obsahem jsou objekty typu X
4. **dict** - slovník, resp asociativní pole

Všechny ostatní datové typy, struktury a objekty jsou jen odvozeny od těchto 4 základních.

5.4 Základní objekt

Základní objekt si lze představit jako jakýsi kontejner, který je základem každé zasílané zprávy. Vychází z něj požadavek i odpověď a má přesně danou strukturu. Jako celek odpovídá datovému typu dict.

Následující tabulka popisuje jednotlivé položky základního objektu, definuje jejih velikosti, datové typy i to, zda-li je povinný při každé zprávě. Velikostí je myšlen počet znaků, respektive počet cifer u číselného datového typu.

Položka	Datový typ	Velikost	Povinna
version	int	2	ano
type	str	10	ano
id	int	11	ano
command	str	20	ne
status	str	3	ne
device_ip	str	15	ano
modules	list(str)		ne
{module-name}	list({dict})		ne

Tabulka 3: Formát požadavku vlastního protokolu

Aby bylo možno lépe specifikovat jednotlivé položky a vysvětlit jejih význam, lze je doplnit o následující podrobnosti:

- **version**
  - **význam:** verze protokolu
  - **příklad:** 1

  Verze protokolu bude vždy zpětně kompatibilní.

- **type**
  - **význam:** typ požadavku
  - **příklad:** "request"
  - **dostupné hodnoty:** "request", "response"

  Každý požadavek obsahuje "request"a naopak každá odpověd obsahuje "response"

- **id**
  - **význam:** unikátní identifikátor požadavku
  - **příklad:** 12345678901

  Každý požadavek ma unikátní id. Nemůže se tedy stát, že se zašlou dva různé požadavky se stejným id. Při posílání odpovědi je použito id požadavku na který se odpovídá, aby se docílilo správného spárování odpovědi s požadavkem, v případě, že komunikujeme s více zařízeními najednou.

- **command**
  - **význam:** specifikuje druh požadavku
  - **příklad:** "ping"
  - **dostupné hodnoty:** "ping", "device_info", "login", "apply", "disconnect", "add", "edit", "get", "get", "delete"

  Vyskytuje se pouze u požadavku, u kterého je povinný.

- **status**
  - **význam:** určuje stav odpovědi
  - **příklad:** "err"
  - **dostupné hodnoty:** "ok", "err"

  Je povinně součástí každé odpovědi a určuje, zda-li byl zaslaný požadavek vykonán správně, nebo nastala-li chyba při jeho vykonávání.

- **device_ip**
  - **význam:** IPv4 adresa zařízení
  - **příklad:** "192.168.1.1"
  - **formát:** "X.X.X.X", kde X je číslo od 0 do 255

  Slouží ke specifikaci, na které zařízení se má dotaz provést. Tato položka je důležitá zejména u konektorů, které komunikují s více zařízeními.

- **modules**

  - **význam:** seznam modulů, kterýh se požadavek týká
  - **příklad:** ["mac-filter", "traffic-shaping"]
  - **dostupné hodnoty:** jména modulů, aktálně "mac-filter", "traffic-shaping"a "firewall".

  Ke každému jménu modulu musí být v požadavku záznam se stejným jménem jako klíčem. Nelze použít jména, která jsou použita pro názvy existujících položek požadavku (například "command")

- **{module-name}**

  - **význam:** obsahuje detail
  - **příklad:** ["username": "uzivatel1", "password":"heslo2
    ]
  - **dostupné hodnoty:** objekty jednotlivých modulů

  Jedná se o slovník, tedy asociativní pole, do kterého jsou převedeny objekty daného modulu.

5.5 Objekty modulů

Z požadavku na dodatečné přidání nových objektů vychází modulární princip všech objektů. Pro jednotlivé typy objektů je zapotřebí tedy definovat moduly.

Modul je vždy uložen jako dict, jehož obsah se shoduje s obsahem objektu, který reprezentuje.

Pro základní komunikaci byly navrhnuty následující moduly, které jsou vždy protokolem podporovány:

5.5.1 login

Pomocí tohoto modulu jsou přenášeny přihlašovací údaje

Seznam položek modulu:

Položka	Datový typ	Velikost	Povinna
username	str	255	ano
password	str	255	ano

Tabulka 4: Formát modulu login

Dodatečné informace o položkách:

- **username**

  - **význam:** uživatelské jméno
  - **příklad:** "admin"

Uživatelské jméno potřebné pro přihlášení

- **password**
  - **význam:** heslo
  - **příklad:** "tajneheslo"

  Heslo, které je zapsáno v podobě prostého textu (plain-text)

5.5.2 device-info

Aby bylo možné zjistit některé základní informace o systému v síťovém prvku, obsahuje tento modul jejich seznam a definici.

Seznam položek modulu:

Položka	Datový typ	Velikost	Povinna
uptime	int	11	ano
system	str	255	ne
cpu-count	int	3	ne
cpu	str	255	ne
cpu-freq	int	5	ne
datetime	int	11	ne
if-count	int	3	ne
ifX-name	str	32	ne
ifX-mac	str	17	ne
ifX-plug	int	1	ne

Tabulka 5: Formát modulu device-info

Dodatečné informace o položkách:

- **uptime**
  - **význam:** Doba běhu systému od posledního spuštění v sekundách
  - **příklad:** 3600

  Doba se zaokrouhluje na celé sekundy

- **system**
  - **význam:** Verze a typ systému
  - **příklad:** "Mikrotik RB750"

  Každý systém si může doplnit vlastní řetězec, který by však měl specifikovat HW resp SW

- **cpu-count**

  - **význam:** Počet dostupných procesorů
  - **příklad:** 2

  Počet jader, kterými disponuje prvek.

- **cpu**
  - **význam:** Typ procesoru
  - **příklad:** "Intel(R) Celeron(TM) CPU"
- **cpu-freq**
  - **význam:** Frekvence, na kterou je taktpván procesor v MHz
  - **příklad:** 1400
- **datetime**
  - **význam:** Datum a čas
  - **příklad:** 1234567890
  - **formát:** Unixtimestamp - počet sekund od 1.1.1970
- **if-count**
  - **význam:** Počet dostupných síťových rozhraní
  - **příklad:** 5 V závislosti na počtu existují také záznamy ifX-* , kde X je číslo z intervalu 0 az (if-count -1)
- **ifX-name**
  - **význam:** Název rozhraní, které je X-té v pořadí.
  - **příklad:** "eth1"
- **ifX-mac**
  - **význam:** MAC adresa, kterou má X-té rozhraní
  - **příklad:** "00:11:22:33:44:55"
  - **formát:** "XX:XX:XX:XX:XX:XX", kde X je šestnáctkové číslo
- **ifX-plug**
  - **význam:** Značí, zda-li rozhraní připojeno
  - **příklad:** 0
  - **formát:** číslo int v rozdahu 0 = False a 1 = True

Tento modul lze použít pouze pro čtení hodnot. Pokusy o zápis nebo úpravu budou ignorovány.

5.5.3 mac-filter

Tento modul slouží k výměně informací o statických ARP záznamech
Seznam položek modulu:

Položka	Datový typ	Velikost	Povinna
ip	str	15	ano
mac	str	17	ano
comment	str	255	ne

Tabulka 6: Formát modulu mac-filter

Dodatečné informace o položkách:

- **ip**
  - **význam:** IPv4 adresa
  - **příklad:** "192.168.1.1"
  - **formát:** "X.X.X.X", kde X je číslo od 0 do 255

  IP adresa, ke které se má nastavit statický ARP záznam
- **mac**
  - **význam:** MAC adresa
  - **příklad:** "00:11:22:33:44:55"
  - **formát:** "XX:XX:XX:XX:XX:XX", kde X je číslo v šestnáctkové soustavě (0 až F)

  MAC adresa zapsána jako text.
- **comment**
  - **význam:** Komentář
  - **příklad:** "honzíkův pc"

  Komentář, který se může objevit jako poznámka u statického záznamu v nastavovaném zařízení

5.5.4 traffic-shaping

Pro potřeby omezování rychlostí jednotlivýh IP adres na směrovačích je navrhnut tento modul.
Seznam položek modulu:
Dodatečné informace o položkách:

- **ip**

Položka	Datový typ	Velikost	Povinna
ip	str	15	ano
down	int	8	ano
up	int	8	ano
comment	str	255	ne

Tabulka 7: Formát modulu traffic-shaping

  - **význam:** IPv4 adresa
  - **příklad:** "192.168.1.1"
  - **formát:** "X.X.X.X", kde X je číslo od 0 do 255

  IP adresa, ke které se má nastavit statický ARP záznam

- **down**
  - **význam:** Rychlost stahování dat v kbps
  - **příklad:** 10000
- **up**
  - **význam:** Rychlost odesílání dat v kbps
  - **příklad:** 2000
- **comment**
  - **význam:** Komentář
  - **příklad:** "honzíkův pc - 10/2 Mbps"

  Komentář, který se může objevit jako poznámka u statického záznamu v nastavovaném zařízení

5.5.5 firewall

Aby bylo možné definovat jednotlivá pravidla ve firewallu, poskytuje tento modul možnost takováto data zaznamenat
Seznam položek modulu:
Dodatečné informace o položkách:

- **chain**
  - **význam:** Sekce firewallu
  - **příklad:** "forward"
  - **dostupné hodnoty:** "input", "output", "forward", "pre", "post"

Položka	Datový typ	Velikost	Povinna
chain	str	3	ano
protocol	str	8	ano
order	int	5	ano
source_ip	str	255	ne
destination_ip	str	255	ne
source_if	str	32	ne
destination_if	str	32	ne
action	str	32	ano
action_def	str	255	ne
comment	str	255	ne

Tabulka 8: Formát modulu firewall

Při použití NATu zůstává nastaveno pouze PRE pro Prerouting a POST pro Postrouting. Ostatní sekce odpovídají sekcím filteru firewallu.

- **protocol**
  - **význam:** Udává pro jaký protokol pravidlo platí
  - **příklad:** "tcp"
  - **dostupné hodnoty:** "tcp", "udp", "all"
- **order**
  - **význam:** Vyj adřuje prioritu v příslušné sekci
  - **příklad:** 0
  - **formát:** číslo od 0 do 65000

  Nižší číslo vždy znamená pozici blíže začátku prohledání filtrů
- **source_ip**
  - **význam:** Zdrojová adresa a port
  - **příklad:** "192.168.0.1"
  - **formát:** [!]ip[/subnet][:port1[-port2]]

  Na začátku lze použít vykřičník pro negování parametru. IP adresa může být doplněna o bitový prefix podsítě. Dále je možné nastavit port za dvojtečku, resp rozsah portů oddělených pomlčkou.
- **destination_ip**
  - **význam:** Cílová adresa a port
  - **příklad:** "!192.168.1.0/24:20-21"

  - **formát:** [!]ip[/subnet][:port1[-port2]]

  Popis položky je totožný s předcházející.

- **source_if**
  - **význam:** Vstupní rozhraní
  - **příklad:** "eth1"
  - **formát:** [!]název-rozhraní

  Na začátku lze použít vykřičník pro negování parametru.

- **destination_if**
  - **význam:** Odchozí rozhraní
  - **příklad:** "!eth1"
  - **formát:** [!]název-rozhraní

  Na začátku lze použít vykřičník pro negování parametru.

- **action**
  - **význam:** Akce, která se má provést při splnění pravidla
  - **příklad:** "drop"
  - **dostupné hodnoty:** "snat", "dnat", "masquerade", "redirect", "accept", "drop", "reject"

  Nastavuje akci, resp cíl pravidla. Pokud je nastaven SNAT, DNAT nebo REDIRECT, je potřeba specifikovat v následujícím poli detaily definice.

- **action_def**
  - **význam:** Rozšiřující definice akce
  - **příklad:** "192.168.1.1:8080"
  - **formát:** ip1[-ip2][:port1[-port2]] nebo port2[:port2] nebo libovolný text

- **comment**
  - **význam:** Komentář
  - **příklad:** "cizí ip z naseho subnetu - zahodit"

  Komentář, který se může objevit jako poznámka u statického záznamu v nastavovaném zařízení

5.6 Příklady komunikace

Ze všeho nejdříve, ihned po přihlášení k serveru, je nutné přihlásit se pomocí uživatelského jména a hesla. Výpis takovéto komunikace v nezašifrované podobě by mohl vypadat například takto:

```
#pozadavek na prihlaseni
{'command': 'login',
 'id': 64042771358L,
 'ip': '127.0.0.1',
 'login': {'password': 'abcd1234', 'username': 'admin'},
 'modules': ['login'],
 'type': 'request',
 'version': 1}

#odpoved, pokud bylo prihlaseni neuspesne
{'status': 'err',
 'version': 1,
 'type': 'response',
 'id': 64042771358}

#odpoved pokud bylo prihlaseni uspesne

{'status': 'ok',
 'version': 1,
 'type': 'response',
 'id': 64042771358}
```

Pokud došlo k autentizaci heslem, můžeme požádat o výpis informací o zařízení:

```
#pozadavek o zaslani systemovych informaci
{'command': 'device-info',
 'id': 94042886585L,
 'ip': '127.0.0.1',
 'type': 'request',
 'version': 1}

#odpoved
{'cpu': 'Intel(R) Core(TM)2 Duo CPU     E7200  @ 2.53GHz',
 'cpu-count': 2,
 'cpu-freq': 2533,
 'datetime': 1399428865,
 'id': 94042886585L,
```

```
'if-count': 1,
'if0-mac': '1c:6f:65:22:5c:b6',
'if0-name': 'eth0',
'if0-plug': 1,
'status': 'ok',
'system': 'Linux version 2.6.32-26-pve (root@lola) (gcc version 4.7.2 (Debian
'type': 'response',
'uptime': 1083459,
'version': 1}
```

6 Implementace

6.1 Databázová vrstva

Pro potřeby komunikace s databází byl navrhnut a implementován balíček zajišťující základní operace nad databází.

```
database/
    init__.py
    mysql.py
model
        __init__.py
        db.py
```

Databázový model lze definovat v adresáři `model`. Každou tabulku reprezentuje právě jeden soubor s konovkou .py, ve kterém je definice třídy, která odpovídá tabulce v databázi. Uvnitř souboru musí být import třídy db, ze které třída nové tabulky dědí.

Třída musí obsahovat `\_table\_name`, které určuje přesný název tabulky v databázi. Sloupce tabulky definujeme podle předpisu `ATTR\_{py-nazev} = "{db-nazev}"`, kde `{py-nazev}` definuje jméno atribudu v databázovém modelu a `{db-nazev}` definuje název sloupce v databázi.

Pokud se jedná o cizí klíč, je potřeba přidat vedle atributu `ATTR` také atribut `FK`, který specifikuje název třídy odpovídající tabulce, na kterou cizí klíč odkazuje.

Nakonec je ještě potřeba naplnit atributy py-nazev výchozími hodnotami podle typu a u cizích klíčů doplnit py-nazev_id = -1.

Představme si, že máme tabulku s názvem „temp“, obsahující následující sloupce:

- id_temp - primární index
- name - textová hodnota
- another_fk - číselná hodnota, která je cizím klíčem odkazující na tabulku "another"

Pro takovou tabulku vytvoříme soubor Temp.py:

```
from db import db

class Temp(db):

    #definice nazvu tabulky
    _table_name ="temp"

    #definice nazvu sloupcu
    ATTR_id = "id_temp"
    ATTR_name = "name"
    ATTR_another = "another_fk"

    #definice nazvu tridy v databazovem modelu pro tabulku ciziho klice
```

```
FK_another = "Another"

  #prirazeni vychozich hodnot
id = 0
name = ""
another = None
another_id = -1
```

Výpis 3: Ukázka definice databázového modelu

S databázovou vrstvou se poté pracuje pomocí třídy mysql.py, která obsahuje následující veřejné metody:

- **__init__**(hostname, username, password, database, prefix)
  Konstruktor celé třídy. Jako parametry jsou údaje o databázovém serveru.

- **select**(table, id, deep, max_deep, where, limit, where_sql)
  Provede dotaz na databázovou tabulku `table`. Pokud je nastaveno `id`, provede se načtení pouze onoho jednoho prvku. Parametry `deep` a `max_deep` nastavují aktuální, resp maximální povolenou hloubku rekurzivního zanoření. Chceme-li vybrat pouze prvky odpovídajíí nějaké podmínce, nastavíme ji pomocí slovníku a vložíme do `where`. `Limit` nám určuje maximální počet prvků, které chceme načíst a `where_sql` slouží pro ruční specifikování podmínek dotazu.

  Na výstupu je vrácen seznam instancí modelu dotazované tabulky.

- **insert**(my_model)
  Vloží do databáze objekt, který je určen instancí v parametru `my_model` a na výstupu vrátí identifikátor právě vytvořeného záznamu.

- **update**(my_model)
  Upraví v databázi objekt, který je určen instancí v parametru `my_model`

- **delete**(my_model)
  Z databáze smaže záznam, který je určen instancí v parametru `my_model`

```
#pouziti unicode v textu
from __future__ import unicode_literals
#import smotne databazove vrstvy
import database

#pripojeni se k databazi se specifikovanymi udaji serveru
sql = database.mysql(hostname='localhost', username='user', password='pass', database='db')

#ziskani 1 radku z tabulky another, ktera ve sloupecku "atr" ma retezec "neco"
a = sql.select("Another",where={'atr':'neco'}, limit=1)[0]

#vytvoreni noveho zaznamu Temp
t = database.model.Temp.Temp()
#nastaveni ciziho klice
t.another = a
```

```
t.another_id = a.id
#nastaveni atributu – sloupecku
t.name = 'nejaky text'

#vlozeni radku do databaze
sql.insert(t)
```

Výpis 4: Ukázka použití databázové vrstvy

6.2 Jádro

Za jádro systému lze považovat všechno, co přímo souvisí s komunikací mezi centrálním serverem, na kterém běží uživatelské rozhraní a jednotlivými síťovými prvkym pomocí vlastního protokolu.

Soubory tvořící přímo jádro systému:

- **server.py** - Naslouchá na straně konektoru
- **formatters.py** - Soubor tříd sloužících k serializaci a deserializaci objektů na formát definovaný vlastním protokolem.
- **crypters.py** - Soubor tříd, kterými je možné šifrovat data ještě před samotným přenosem.
- **client.py** - Klient pro připojení k serverům běžícím na vzdálených síťových zařízeních.
- **uploader.py** - Zjistí z databáze nastavení a rozešle je všem síťovým prvkům.
- balíček **database** - Poskytuje databázový model na centrálním prvku.

6.3 Konektory

- **database_connector.py** - poskytuje ostatním konektorům přístup k datům uloženým v SQLite databázi.
- **linuxapi.py** - Api a konektor pro práci s lokálním systémem založeným na linuxu.
- **mikrotikapi.py** - Api, konektor a potřebné třídy pro komunikaci se vzdáleným zařízením se systémem RouterOS pomocí Mikrotik API-SSL.

6.4 Webové rozhraní

Kvůli snadné obsluze bylo rozhodnuto, že vizuální prostředí bude napsáno jako webová aplikace v JavaScriptu a bude plně dynamická, aby se prezentovala schopnost tvorby komplexnějších systémů v kontextu prohlížeče.

Jako ideální framework pro rozsáhlé uživatelské rozhraní vyhrál balík DojoToolkit ve verzi 1.9. Obsahuje množství ovládacích prvků a jeho činnost je vyladěna na většině moderních internetových prohlížečů.

Celé uživatelské rozhraní je generováno čistým JavaScriptem pomocí balíčku dijit, který je součástí frameworku DojoToolkit

6.4.1 Přístup k datům

Pro komunikaci mezi uživatelským rozhraním a databází byl v Pythonu vytvořen vlastní server HTTP, do kterého byla implementována podpora protokolu RESTful. Python REST server byl pomocí předem napsaného databázového modelu připojen k SQL databázi.

REST server podporuje všechny základní operace nad databází, včetně autorizace pomocí náhodně generovaných klíčů s omezenou časovou platností. Při přihlášení je vygenerován unikátní klíč, obdoba sessionid a při každém dotazu na server je zjištěno, zda-li byl požadavek odeslán od přihlášeného uživatele a jestli nevypršela doba jeho sezení. Pokud je ověření v pořádku, automaticky se prodlouží doba sezení o stanovenou dobu.

Na straně JavaScriptu bylo využito knihovny JsonRest, která je součástí balíčku dojo, stejnojmeného frameworku.

7 Testování

7.1 Nastavení

Pro spuštění musí být systém nakonfigurován

7.1.1 Nastavení jádra

Jako první věc je potřeba vytvořit v MySQL novou databázi „jbednar“ a uživatele a přidělit mu oprávnění k nové databázi. Pro vytvoření struktury použijeme skript `create_database.sql`

V souboru `/web-backend/config.ini` nastavíme potřebné údaje pro spojení s databází. Nastavíme rovněž absolutní cestu k adresáři, kde se nachází souboru `client.py`. Tuto cestu nastavíme v sekci client, hodnota path. Ve stejné sekci nastavíme cestu k certifikátu, který bude použit pro komunikaci s jednotlivými prvky.

Server spustíme příkazem `python server.py`

7.1.2 Nastavení konektorů

7.1.2.1 Mikrotik Konektor V podadresáři `certs` nalezneme veřejný a privátní klíč pro komunikaci s centrálním serverem.

V souboru `mikrotiks.cfg` nastavíme cestu k cerfifikátům pro jednotlivé adresy, se kterými se komunikuje

Spustíme skript na vygenerování nové databáze pro konektor. V /web-backend spustíme `python create_db.py NAZEV`, kde NAZEV je pozadovany nazev souboru. Pro každé zařízení, se kterým Konektor komunikuje je potřeba vytvořit zvlášť databázi ve tvaru IP.db (například 192.168.1.1.db) a tu nahrát do adresáře s konektorem.

Konektor na libovolném počítači s linuxem spustíme pomocí příkazu `python main.py`

7.1.2.2 Linux Konektor V podadresáři `certs` nalezneme veřejný a privátní klíč pro komunikaci s centrálním serverem.

V souboru `linux_connector.cfg` si v sekci `login` nadefinujeme jméno a heslo pro přihlášení ke Konektoru a v sekci `database` upravíme parametr `host`, do kterého zapíšeme požadované jméno databáze. Tento soubor musí existovat. Lze jej vytvořit například podle výše zmíněného postupu

Konektor na zvoleném počítači s linuxem spustíme pomocí příkazu `python main.py`

7.1.3 Nastavení uživatelského rozhraní

Libovolný webservere nastavíme tak, aby otevřel adresář `www` a soubor index.html. Chceme-li omezit přístup k aplikaci můžeme tak učinit například pomocí direktivy .htaccess AuthType, tedy zapnout výchozí HTTP ověření.

Poté, co otevřeme web se přepneme na záložku Definice. Zde je potřeba nadefinovat základní prvky, ze kterých budem skládat zařízení. V jednotlivých podzáložkách je potřeba nastavit

- **Typy rozhraní** - vytvořit alespoň jeden typ rozhraní
- **Konektory** - vytvořit dva konektory. Jeden bude mít nastaven název „Mikrotik“ a druhý „Linux“.
- **Typy zařízení** - Zde si definujeme zařízení, která poižíváme v sítí a nastavíme jim potřebné konektory.
- **Typy rozhraní** - vytvořit alespoň jeden typ rozhraní
- **Typy linek** - je potřeba vytvořil alespoň jeden typ linky

Následně se přepnem zpět na záložku Zařízení. Zde musíme vytvořit potřebný počet zařízení.

U každého zařízení je potřeba nastavit:

- **Jméno zařízení** - jméno, které se u něj zobrazuje
- **Typ zařízení** - zvolit o jaké zařízení se jedná
- **IP adresa** - IP adresa, kterou má zařízení přiřazenou
- **Umístění** - Vytvořit si adresu a tu následně zvolit
- **Aktivní** - Zvolit prvek jako aktivní
- **IP konektoru** - IP adresa, na které běží jádro konektoru
- **Uživatel** - Uživatelské jméno pro vstup do zařízení. V případě Linuxu se jedná o přihlašovací údaje vyplněné v souboru linux_connetor.cfg
- **Heslo** - Má stejný vztah jako Uživatel
- **Zobrazeno** - Vybrat, že má být zařízení viditelné na mapě
- **Pozice X, Y** - Určuje pozici, kde je router zobrazen v diagramu
- **Poznámky** - volitelné poznámky k zařízení

Po úspěšném vložení se zobrazí směrovač v mapě a lze s ním pohybovat. Pozor, pozice v mapě se ukládá při každém přesunu značky routeru.

Následně lze mezi více routery nastavit vazby pomocí „Upravit linky“. Při vybrání některého ze zařízení je možno zvolit „Upravit pravidla“

V této části se definují vlastní pravidla pro firwall, omezení rychlosti a filtrování MAC adres.

Ve chvíli, kdy jsou potřebná pravidla nastavena je možno okno zavřít a tlačítkem „Nahrát konfiguraci“ přesunout pomocí vlastního protokolu pravidla na cílové zařízení automaticky použít nastavení.

8 Závěr

Výsledkem práce je návrh a plně funkční implementace vlastního protokolu pro distribuci pravidel mezi síťové prvky, plně dynamické webové rozhraní, včetně komunikace s vlastním serverem poskytujícím data a modulární systém pro přenos dat mezi tímto webovým rozhraním a cílovými prvky.

Pro realizaci bylo zapotřebí prozkoumat dostupné možnosti vzdálené správy síťových prvků s uživatelským rozhraním, nastudovat a porozumnět komunikaci pomocí speciálního API se zařízeními firmy Mikortik.

Realizace této diplomové práce pro mne byla přínosná hlavně z hlediska komunikace mezi různými systémy, pro mne prvním praktickým užitím mechanismu TLS a problematikou digitálních certifikátů. Velice poučné je také použití architektury REST.

Jelikož je mi tato problematika, jako správci malé sítě, velice blízká, plánuji v budoucnu na tuto práci navázat a vytvořit plně automatizovaný systém, který umožní zařízení nejen nastavovat, ale také sledovat. Pro základ takovéto aplikace bude vhodné využít v této práci navrhnutý protokol.

9 Reference

[1] ŠVEC, Jan, *Seriál Python - Létající cirkus* Root.cz : informace nejen ze světa Linuxu [online]. Verze 1.0. 2001 [cit. 2014-05-05]. Létající cirkus. Dostupné z WWW: http://www.root.cz/clanky/letajici-cirkus/

[2] DIERKS, T., RFC 5246. *The Transport Layer Security (TLS) Protocol: Version 1.2*. USA: The IETF Trust, 2008. Dostupné z: http://www.ietf.org/rfc/rfc5246.txt

[3] MALÝ, Martin., REST: architektura pro webové API. *Zdroják* [online]. 2009 [cit. 2014-05-06]. Dostupné z: http://www.zdrojak.cz/clanky/rest-architektura-pro-webove-api/

[4] BRYAN, P. , RFC 6901. *JavaScript Object Notation (JSON) Pointer*. USA: Internet Engineering Task Force, 2013. Dostupné z: http://www.ietf.org/rfc/rfc6901.txt

[5] BERNERS-LEE, Tim., RFC 3986. *Uniform Resource Identifier (URI): Generic Syntax*. USA: The Internet Society, 2005. Dostupné z: http://www.ietf.org/rfc/rfc3986.txt

[6] PURDY, Gregor N. *Linux iptables Pocket Reference: Firewalls, NAT Accounting.* USA: OReilly Media, 2004, 96 s. ISBN 978-0-596-00569-6.

[7] LEBEDA, Martin, SQLite: ultra lehké sql. *Root.cz* [online]. 2003 [cit. 2014-05-06]. Dostupné z: http://www.root.cz/clanky/sqlite-ultra-lehke-sql/

[8] FIELDING, Roy., RFC 2616. *Hypertext Transfer Protocol: HTTP/1.1*. USA: The Internet Society, 1999. Dostupné z: http://www.ietf.org/rfc/rfc2616.txt

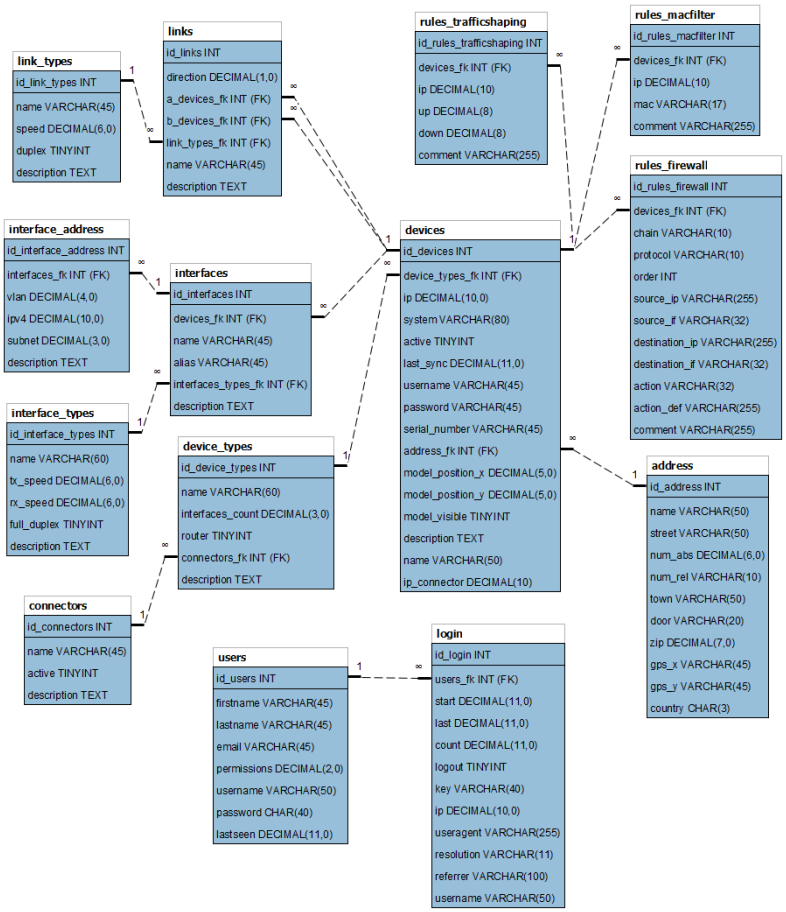
[9] *SQLite: Documentation* [online]. 2014 [cit. 2014-05-06]. Dostupné z: http://www.sqlite.org/docs.html

[10] *Learn REST: A RESTful Tutorial*[online]. 2012 [cit. 2014-05-06]. Dostupné z: http://www.restapitutorial.com

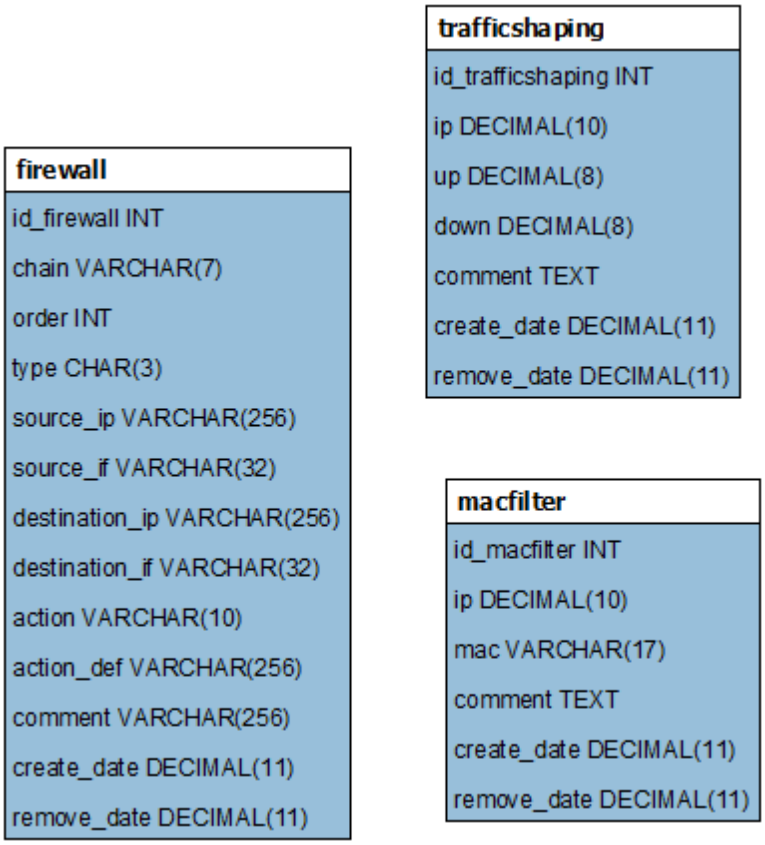
[11] RODRIGUEZ, Alex., IBM Developer Works: RESTful Web services: The basics. IBM. *RESTful Web services: The basics* [online]. 2008 [cit. 2014-05-06]. Dostupné z: https://www.ibm.com/developerworks/webservices/library/ws-restful/

A Diagramy a obrazovky uživatelského rozhraní

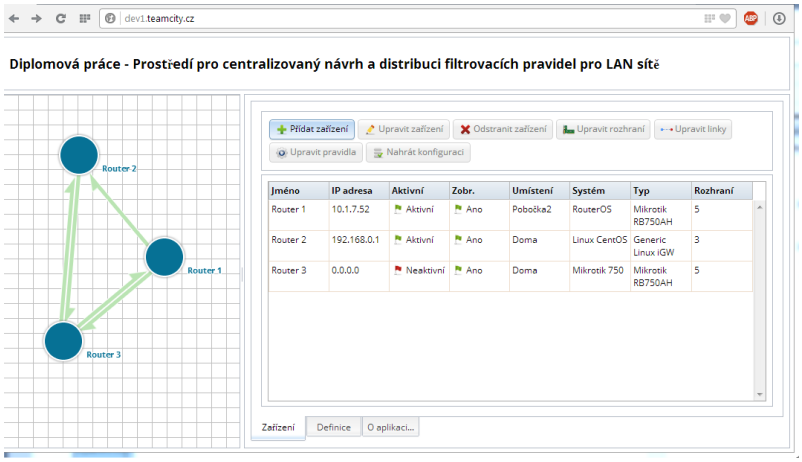
V této části se nachází diagramy popisující chování programu jako celku. Kvůli jejich obecnosti a velikosti byl přesunuty do speciální části v přílohách.



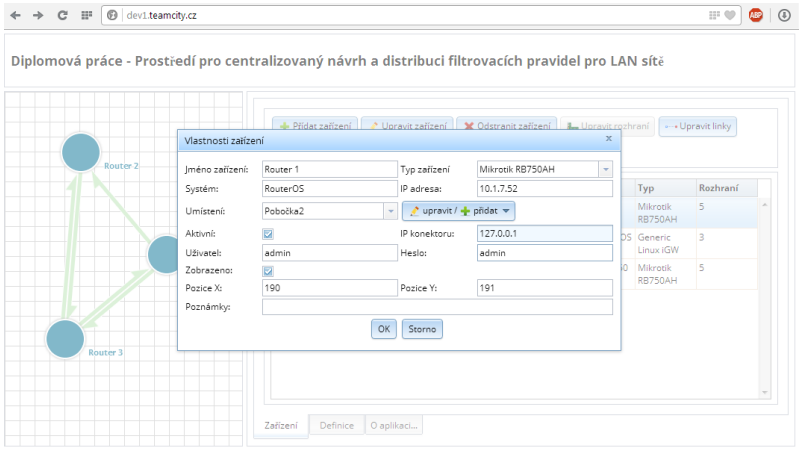
Obrázek 7: ER diagram databáze pro jádro systému



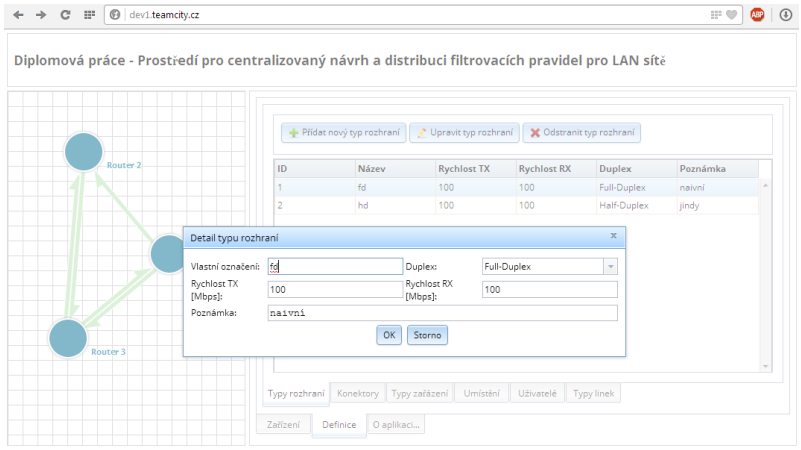
Obrázek 8: ER diagram databáze pro konektor zařízení



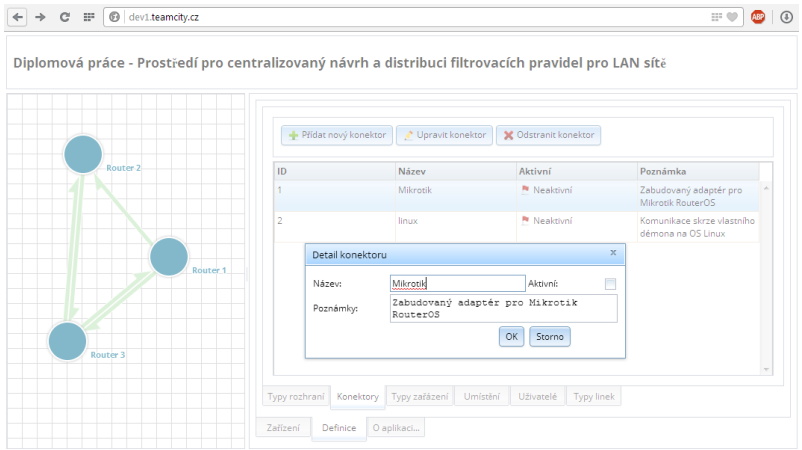
Obrázek 9: Hlavní okno po přihlášení



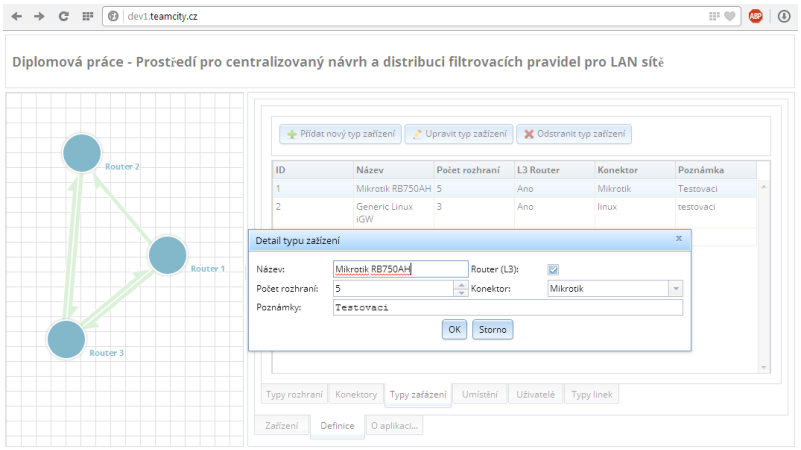
Obrázek 10: Nastavení zařízení



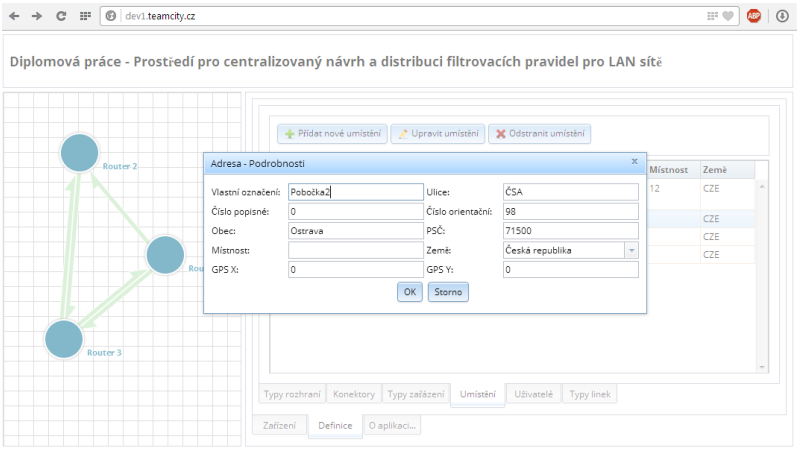
Obrázek 11: Definice typu rozhraní



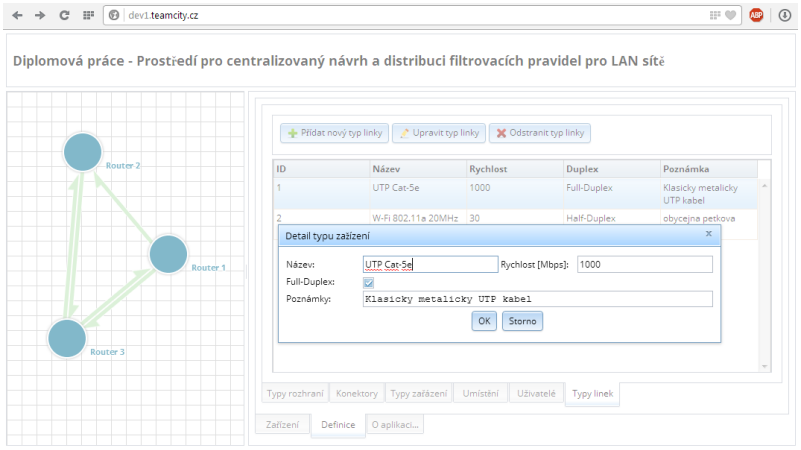
Obrázek 12: Natavení Konektoru



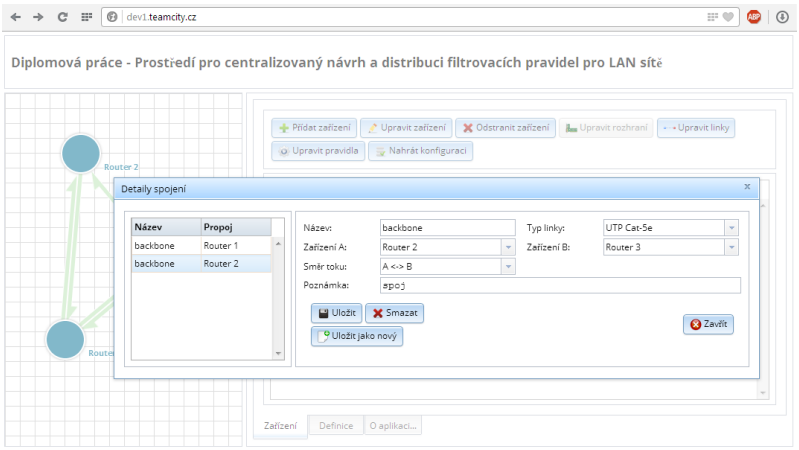
Obrázek 13: Definice typu zařízení



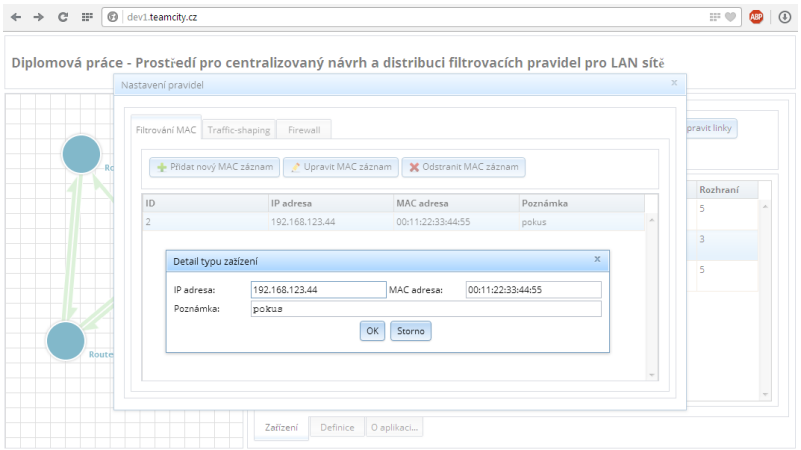
Obrázek 14: Nastavení adresy - umístění



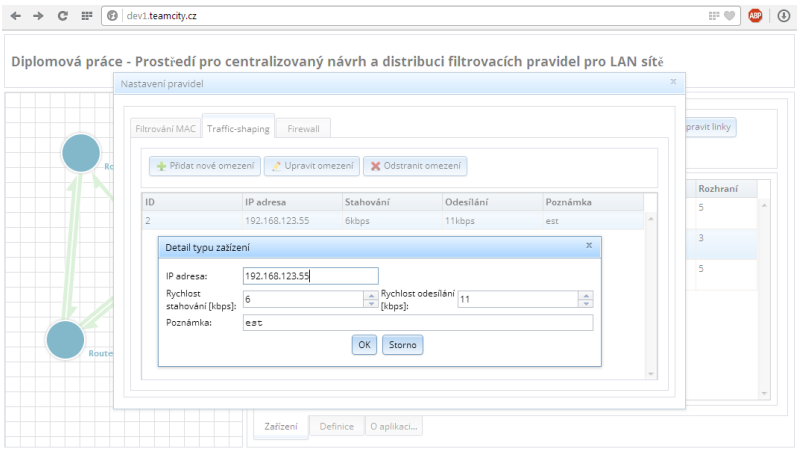
Obrázek 15: Definice linek



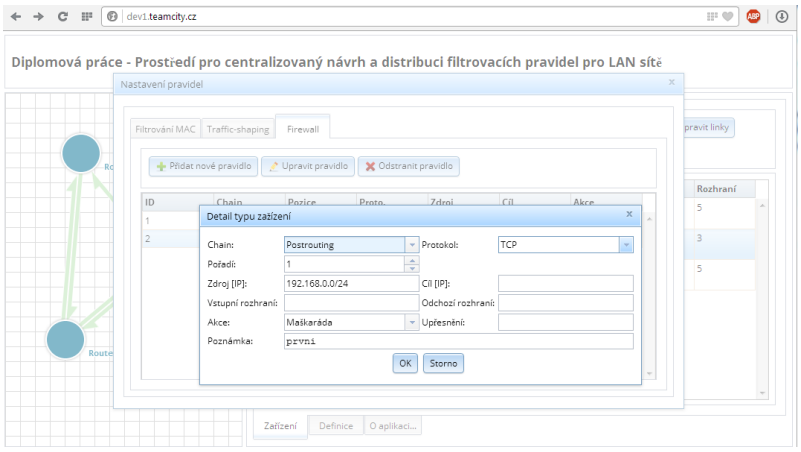
Obrázek 16: Nastavení spojení routerů linkami



Obrázek 17: Nstavení MAC Filtrace



Obrázek 18: Nastavení traffic-shapingu



Obrázek 19: Nastavení pravidel firewallu